Reqtrace AI: A Hybrid Generative AI And Knowledge Graph Framework For Software Requirement Mutation Detection And Impact Analysis

Dr. C. VidyaRaj[1], Ms.Deepashree M[2], Vikas[3], Vivek B Akki[4]
[1]*Professor, Department of Computer Science and Engineering NIE, Mysuru*
[2]*Assistant Professor, Department of Computer Science and Engineering NIE, Mysuru*
[3,4]*Department of Computer Science and Engineering NIE ,Mysuru*

***Abstract*—Software requirements can change several times during the development of a software system. Tracking these changes across different versions of Software Requirement Specification documents manually can be difficult and time-consuming, especially when the number of requirements increases. This paper presents ReqTrace AI, a framework developed to automate requirement extraction, change detection, and traceability. The system processes SRS documents and uses Gemini AI to identify and structure individual requirements. Requirements from different versions are then compared using Sentence Transformer embeddings and cosine similarity to identify Added, Deleted, and Modified requirements based on their meaning. The detected requirements and their relationships are stored in a Neo4j Knowledge Graph, which is used to trace changes and identify related requirements and potentially affected software components. A confidence-based validation step is also included so that uncertain relationships can be reviewed by the user. The proposed workflow brings requirement extraction, semantic comparison, graph-based traceability, and impact analysis together in a single system. The implementation and testing of the system show that this approach can reduce the manual effort involved in tracking requirement changes and provide a more structured way to analyze the impact of evolving requirements.**

***Index Terms*—software requirements, requirement traceability, requirement mutation detection, semantic similarity, Knowledge Graph, impact analysis, generative artificial intelligence**

I. INTRODUCTION

Software requirements often change during the development and maintenance of a software system. New requirements may be introduced, existing requirements may be removed, and previously defined requirements may be modified as the system evolves. Tracking these changes across different versions of Software Requirement Specification (SRS) documents can become difficult when it is performed manually. As the number of requirements increases, manual comparison can require considerable effort and may result in important changes being overlooked. Understanding how a changed requirement affects related requirements and software components is another challenge in maintaining effective requirement traceability.

Traditional approaches to requirement change tracking commonly use manual comparison, spreadsheets, or traceability matrices. These approaches can help maintain records of requirements, but they provide limited support for automatically identifying changes based on the meaning of requirement statements. Existing research has separately explored areas such as Knowledge Graph-based software knowledge representation, semantic graph querying, program-level change-impact analysis, and the use of Large Language Models for software requirements. Research on requirements traceability has also examined techniques for linking requirements with related software artifacts. These studies provide useful approaches for individual parts of the problem, but there is a need for a workflow that connects requirement extraction, semantic change detection, traceability, and impact analysis.

To address this need, this paper presents ReqTrace AI, an AI-driven framework for analyzing changes in SRS documents and supporting requirement traceability.

The system first processes the input SRS document and uses Gemini AI to identify and structure individual requirements. Requirements from different versions are then converted into semantic representations using Sentence Transformer embeddings. Cosine similarity is used to compare requirements based on their meaning and classify changes as Added, Deleted, or Modified. The identified requirements and their relationships are subsequently represented in a Neo4j Knowledge Graph, providing a structured representation that can be used to trace changed requirements and examine related requirements and potentially affected software components.

The main contributions of this work are fourfold. First, the framework provides AI-assisted extraction and structuring of requirements from SRS documents. Second, it applies semantic similarity to detect requirement mutations between different document versions. Third, it uses a Knowledge Graph to represent requirement relationships and support dynamic traceability. Finally, it uses the identified relationships to support change-impact analysis. A confidence-based validation mechanism is also included so that uncertain relationships can be reviewed by the user. Together, these components provide an integrated workflow for analyzing evolving software requirements.

II. RELATED WORK

A. Knowledge Graphs and Software Knowledge Representation

Knowledge Graphs have been used to represent software-related knowledge and relationships in a structured form. Hochgeschwender *et al.* explored graph-based software knowledge representation using labeled property graphs and Neo4j, demonstrating how software and domain models can be stored and queried semantically [1]. Other research has investigated semantic querying in Neo4j and Knowledge Graph completion techniques, showing the usefulness of graph structures for representing and retrieving connected information [3], [4].

These studies provide the foundation for using Knowledge Graphs in ReqTrace AI. However, their primary focus is on software knowledge representation, graph querying, or graph completion rather than tracking changes between different versions of SRS documents.

B. Requirement Analysis, Traceability, and Change Impact

Research has also explored automated analysis of software requirements and change impact. Li *et al.* proposed a software-network-based approach for analyzing how program changes propagate through software systems [2]. Research on Large Language Models has further shown that LLMs can assist with software requirement quality assessment and improvement [5]. A systematic literature review by Mucha *et al.* examined requirement traceability techniques and highlighted the importance of maintaining relationships between requirements and related software artifacts [6].

These studies address important parts of requirement engineering, but they generally focus on individual aspects such as program-level change analysis, requirement quality assessment, or traceability. ReqTrace AI combines these concepts into a single workflow by extracting requirements from SRS documents, detecting Added, Deleted, and Modified requirements using semantic similarity, representing requirement relationships through a Neo4j Knowledge Graph, and supporting impact analysis of changed requirements.

III. PROPOSED METHODOLOGY

ReqTrace AI follows a sequential workflow for extracting, comparing, and tracing software requirements across different versions of Software Requirement Specification (SRS) documents. The proposed methodology combines PDF processing, Generative AI, semantic similarity analysis, and Knowledge Graph-based relationship modeling.

A. SRS Document Processing and Requirement Extraction

The process begins with the upload of an SRS document in PDF format. The document is parsed to extract its textual content and prepare it for further analysis. Gemini AI is then used to identify individual requirement statements and organize them into a structured format with requirement identifiers and categories. This step converts the unstructured SRS

document into a set of requirements that can be processed by the subsequent modules.

B. Semantic Requirement Comparison

Requirements from different versions of the SRS are compared using Sentence Transformer embeddings. Each requirement is converted into a numerical representation that captures its semantic meaning. Cosine similarity is then used to measure the similarity between requirements from the earlier and newer versions. Based on the similarity and comparison process, requirements are classified as Added, Deleted, or Modified. This allows the system to identify meaningful changes rather than relying only on exact keyword matching.

C. Knowledge Graph Construction

The extracted requirements and their relationships are represented using a Neo4j Knowledge Graph. Requirements are stored as graph nodes, while relationships between requirements and related software components are represented as edges. Relationship confidence and supporting information are used to assist in validating the detected connections. The graph provides a structured representation of requirement dependencies and enables the system to trace changes through related elements.

D. Impact Analysis and Validation

After requirement changes and relationships have been identified, the system performs impact analysis to determine which related requirements and software components may be affected by a change. A confidence-based validation mechanism is used to categorize detected relationships for acceptance or further review. This allows uncertain relationships to be examined by the user before they are used for final traceability analysis.

E. Overall Workflow

The complete workflow can be summarized as:

SRS PDF → Text Extraction → Gemini AI Requirement Extraction → Sentence Transformer Embeddings → Cosine Similarity → Added/Deleted/Modified Detection → Neo4j Knowledge Graph → Relationship Validation → Impact Analysis

This integrated workflow allows ReqTrace AI to connect requirement extraction, semantic change detection, Knowledge Graph-based traceability, and impact analysis within a single system.

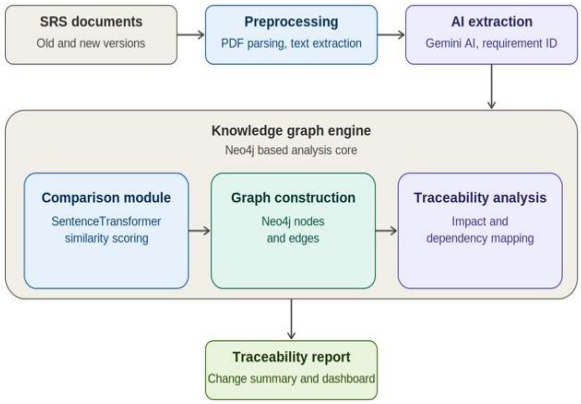

Fig. 1. Overall workflow of the proposed ReqTrace AI framework.

IV. SYSTEM IMPLEMENTATION

ReqTrace AI is implemented as a web-based application that combines document processing, Generative AI, semantic analysis, and Knowledge Graph technologies. The implementation is divided into modules so that each stage of requirement analysis can be performed independently while remaining connected to the overall workflow.

A. Document Processing Module

The document processing module accepts SRS documents in PDF format and extracts their textual content for analysis. PDF parsing and text preprocessing are performed before the extracted content is passed to the requirement extraction stage. This prepares the document for structured analysis and reduces the amount of unprocessed information handled by subsequent modules.

B. Requirement Extraction Module

The requirement extraction module uses Gemini AI to identify individual requirement statements from the processed SRS content. The extracted requirements are organized using unique identifiers and requirement categories. This structured representation provides the input required for comparing requirements across different SRS versions.

C. Requirement Comparison Module

The comparison module uses SentenceTransformer to generate embeddings for the extracted requirements. Cosine similarity is applied to compare requirements

between the earlier and newer versions of the SRS. Based on the comparison, the system identifies requirements as Added, Deleted, or Modified. This semantic comparison allows the system to consider the meaning of requirement statements rather than depending only on matching words.

D. Knowledge Graph Module

The Knowledge Graph module uses Neo4j to store requirements and their relationships. Requirement information is represented as nodes, while relationships between requirements and related software components are represented as graph edges. Relationship confidence and supporting evidence are used during the validation process. The resulting graph provides a structured representation of the connections needed for requirement traceability.

E. Impact Analysis Module

The impact analysis module uses the relationships represented in the Knowledge Graph to trace the effects of changed requirements. The system identifies related requirements and potentially affected software components, allowing the user to examine the possible consequences of requirement changes. A confidence-based validation mechanism allows uncertain relationships to be reviewed before they are used in the final traceability analysis.

F. Application Interface and Technology Stack

The system provides a web-based interface for uploading SRS documents, viewing extracted requirements, comparing document versions, reviewing detected changes, validating relationships, visualizing the Knowledge Graph, and examining impact-analysis results.

TABLE 1 TECHNOLOGIES USED IN REQTRACE AI

Component	Technology
Backend	Python, FastAPI
Frontend	React.js
Requirement Extraction	Gemini AI
Semantic Representation	Sentence Transformer
Similarity Analysis	Cosine Similarity
Knowledge Graph	Neo4j
Document Processing	PDF Parsing

The implementation uses Python and FastAPI for backend services, React.js for the frontend, SentenceTransformer for semantic representation, PDF-processing libraries for document extraction, and Neo4j for Knowledge Graph storage and querying.

V. RESULTS AND DISCUSSION

The implemented ReqTrace AI system was tested using SRS documents to evaluate its ability to extract requirements, identify changes between document versions, construct requirement relationships, and support impact analysis. The results demonstrate the operation of the complete workflow from document upload to final traceability analysis.

A. Requirement Extraction Results

The requirement extraction module successfully processed uploaded SRS documents and generated structured requirement information. In the demonstrated test workflow, the system extracted 50 individual requirements and organized them into categories such as functional, security, and data requirements. The extracted requirements were then made available for further comparison and traceability analysis.

B. Requirement Mutation Detection

The version comparison module was used to compare an older and newer version of an SRS document. In the demonstrated comparison, the system identified 3 Added, 3 Deleted, and 9 Modified requirements. The system also provided a side-by-side view of modified requirements, allowing users to examine the changes in their original context. For example, one modified requirement showed a change related to authentication. The use of semantic similarity allows the comparison process to consider the meaning of requirement statements rather than relying only on exact textual matches. This is useful when a requirement is reworded while retaining related meaning.

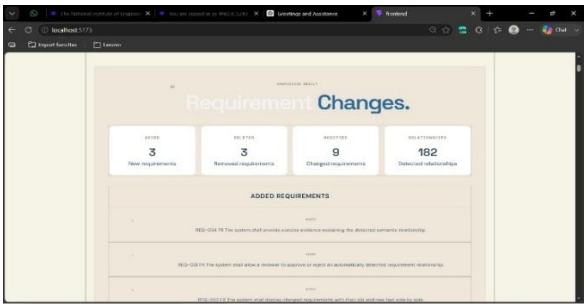

Fig. 2. Summary of detected requirement mutations between SRS versions.

C. Relationship Analysis and Knowledge Graph

The identified requirements and their relationships were represented using a Neo4j Knowledge Graph. In the demonstrated workflow, the system detected 182 relationships during the requirement comparison and traceability process. Relationship confidence was used to support validation, with relationships in the intermediate confidence range being presented for manual review rather than being automatically accepted.

The Knowledge Graph visualization represented 51 requirements connected to 9 software components, providing a structured view of the relationships between requirements and components. This graph representation allows users to examine requirement dependencies and trace relationships through the system.

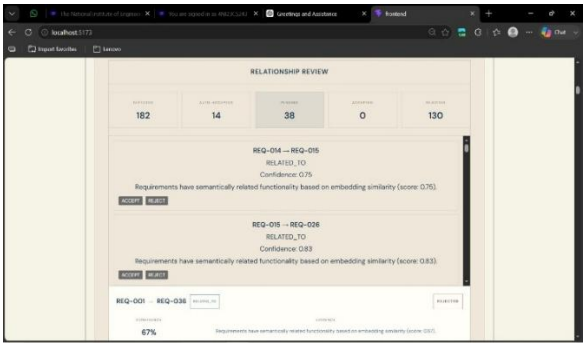

Fig. 3. Confidence-based validation of detected requirement relationships.

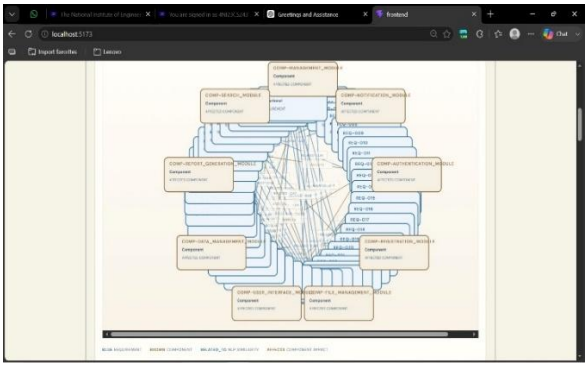

Fig. 4. Knowledge Graph representation of requirements and related software components.

D. Impact Analysis

The impact analysis module uses the relationships stored in the Knowledge Graph to determine the components associated with changed requirements. The system provides a detailed traceability view showing related requirements and potentially affected software components. For example, the demonstrated workflow traces REQ-052 to the User Interface Module, showing how a requirement change can be connected to an affected component.This provides information beyond a simple document-difference operation because the user can examine the relationships associated with a detected requirement change.

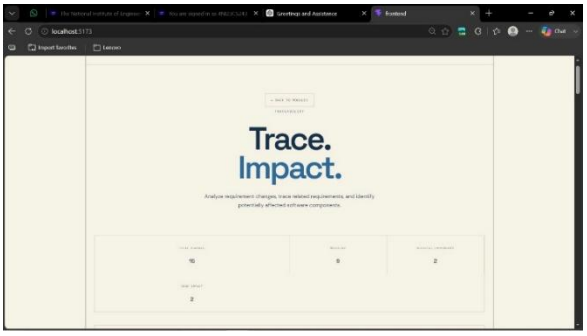

Fig. 5. Impact analysis summary of changed requirements and affected components.

E. System Testing

The system was tested using unit, integration, and system-level testing. The documented test cases included valid document upload, handling of a corrupted or zero-byte PDF, extraction of a document containing 20 requirements, and comparison of two SRS versions. All four documented test cases were marked Pass. System testing also verified the end-to-end workflow, including requirement extraction, comparison, classification, frontend display, and storage of traceability results in Neo4j.

Overall, the results demonstrate that ReqTrace AI can perform the intended workflow of SRS processing, requirement extraction, semantic change detection, Knowledge Graph construction, relationship validation, and impact analysis. The system combines automated processing with human review for uncertain relationships, providing a structured approach to analyzing evolving software requirements.

VI. CONCLUSION AND FUTURE WORK

CONCLUSION

ReqTrace AI demonstrates an AI-driven approach to automating software requirement traceability and change-impact analysis. The system integrates requirement extraction, semantic comparison,

Knowledge Graph modeling, and impact analysis into a single workflow for analyzing changes across different versions of Software Requirement Specification documents.

The implemented system uses Gemini AI to extract and structure individual requirements from SRS documents. SentenceTransformer embeddings and cosine similarity are then used to compare requirements across versions and classify them as Added, Deleted, or Modified. The detected requirements and their relationships are represented using a Neo4j Knowledge Graph, allowing users to examine requirement dependencies and trace changes toward related software components. A confidence-based validation mechanism also provides an opportunity for users to review uncertain relationships. The demonstrated results show that the system can perform the intended requirement extraction, version comparison, relationship analysis, Knowledge Graph construction, and impact-analysis workflow. The testing process further verified important system functions, including document handling, requirement extraction, version comparison, and end-to-end traceability. These results indicate that combining AI-based processing with graph-based representation can provide a structured approach to managing evolving software requirements.

FUTURE WORK

The current implementation can be extended in several directions. Support for additional SRS formats such as DOCX and TXT can make the system applicable to a wider range of development environments. The processing pipeline can also be optimized for larger SRS documents to improve scalability and reduce processing time.

Future versions can extend the Knowledge Graph beyond requirement-to-requirement and requirement-to-component relationships by adding direct traceability to source-code modules and test cases. Integration with development and project-management platforms such as Jira and Azure DevOps could further connect requirement analysis with existing software development workflows.

Further improvements can also explore advanced Large Language Models and domain-specific semantic models for more detailed requirement classification, relationship detection, and impact analysis. Additional dashboards and analytical reports could be introduced to help development teams monitor requirement changes, traceability coverage, and the overall status of requirement analysis.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to The National Institute of Engineering, Mysuru, for providing the academic environment and facilities required to carry out this work. The authors are grateful to Dr. C. VidyaRaj, Professor, Department of Computer Science and Engineering, NIE, Mysuru, for his valuable guidance and support. The authors also sincerely thank Ms. Deepashree M, Assistant Professor, Department of Computer Science and Engineering, NIE, Mysuru, for her continuous guidance, constructive suggestions, and encouragement throughout the development of ReqTrace AI. The authors also thank the faculty members and everyone who supported the successful completion of this work.

REFERENCES

[1] N. Hochgeschwender, S. Schneider, H. Voos, H. Bruyninckx, and G. K. Kraetzschmar, “Graph-based software knowledge: Storage and semantic querying of domain models for run-time adaptation,” in *Proc. IEEE Int. Conf. Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, San Francisco, CA, USA, 2016, pp. 83–90.

[2] S. Li, F. Deng, X. Guo, Z. Sun, and Z. Zou, “Research on program change impact analysis methods based on software networks,” 2024.

[3] N. N. Tun and N. N. Yee, “Comparison between semantic (rules-based) query and keyword-based query in Neo4j graph database,” 2025.

[4] Y. Chen, S. Li, H. Fang, Y. Li, and J. Hu, “A knowledge graph completion model combining graph convolutional networks with self-attention mechanisms,” 2024.

[5] “Leveraging LLMs for the quality assurance of software requirements,” in *Proc. IEEE 32nd Int. Requirements Engineering Conf. (RE)*, 2024.

[6] J. Mucha, A. Kaufmann, and D. Riehle, “A systematic literature review of pre-requirements specification traceability,” 2024.